Beschreibende Datenanalyse

Joern Ploennigs · AI4SC

> KISS - Keep it Simple, Stupid
>
> — Kelly Johnson

Der erste Schritt beim Analysieren eines neuen Datensatzes ist es sich mit den Daten vertraut zu machen. Hierbei ist es noch nicht Ziel aus den Daten Statistiken zu extrahieren, sie zu Visualisieren oder zu Säubern, was wir in separaten Abschnitten behandeln. Stattdessen versucht man sich einen Überblick zu verschaffen: Welche Spalten gibt es? Gibt es fehlende Werte? Wie sind die Einträge verteilt?

Gerade im Bauwesen, z.B. bei Sensor- oder Baudaten, ist das entscheidend, um zu erkennen, ob die Daten überhaupt für eine Modellierung geeignet sind.

Erste Schritte: Kenne deine Daten

Als nächstes machen wir einige Beispiele für explorative Datenanalyse mit Pandas. Hierfür laden wir als Beispiel die Liste aller aktuellen Baustellen in Rostock an, die von der Stadt Rostock im Open Data Portal zur Verfügung gestellt wird. Öffentliche Daten wie diese zeigen reale Bauaktivitäten und die Analyse hilft z.B. bei der Planung, Verkehrsführung oder Risikobewertung bei parallelen Projekten. Solche Daten können später genutzt werden, um Modelle zur Vorhersage von Bauverzögerungen, Verkehrsbelastung oder Personalplanung zu trainieren.

Wir lesen die komprimierte baustdatendatei direkt in Pandas ein. Da diese durch Komma getrennt ist, müssen keinen Trennzeichen angeben, da das Komma der Standard-Trennzeichen ist.

```
import numpy as np # Import von NumPy
import pandas as pd # Import von Pandas

baust  = pd.read_csv("../data/Baustellen/baustellen_flat.csv")
```

Als ersten Schritt können wir die Größe des Datensatzes abfragen. Datenrahmen haben ein Attribut `.shape`, genau wie numpy-Arrays:

```
baust.shape
```

```
(204, 19)
```

Die Antwort sagt uns die Anzahl der Zeilen und Spalten an. Z.B. `(204, 19)` bedeutet 204 Zeilen mit je 19 Spalten. Das hilft abzuschätzen, wie umfangreich der Datensatz ist und ob er vollständig erscheint.

Als nächstes könnten wir einen schnellen Blick auf ein paar Zeilen der Daten werfen. Die Attribute `.head()` und `.tail()` sind nützlich, um sich die ersten und letzten Zeilen anzusehen, `.sample()` extrahiert ein paar zufällige Zeilen:

```
baust.head()
```

	latitude	longitude	[illegible]	spar-te	von	nach	bau-beginn	bau-ende	ver-kehrs-be-ein-traech-ti-gun-gen	bau-mass-nah-me	dau-er
0	54.08[illegible]	[illegible]	[illegible] Rostock, Hanse- und Universitätsstadt [illegible] Rostock, Hanse- und Universitätsstadt [illegible] Budapester Str.	Fernwärmeleitung	21	NaN	2024-[illegible] 07:00[illegible]	[illegible]	[illegible]rungsmaßnahmen entlang der Straße, Verkeh…	FW-[illegible]tung i.a. der Hanseatic	39.375000
1	54.06[illegible]	[illegible]	[illegible] Rostock [illegible] Rostock, Han- [illegible] Rostock, Han- [illegible] Satower Str.	Grünpflege	56	65	2024-[illegible] 07:00[illegible]	[illegible]	[illegible]	Baumpflanzung	24.291667

	la- ti- tu- de	gi- tu- de						spar- te	von bau- be- ginn	nach bau- er- kehrs- be- ein- traech- ti- gun- gen	bau- mass- nah- me	dau- er
			se- und Uni- ver- si- täts- stadt	se- und Uni- ver- si- täts- stadt						ge Sper- rung, Si- che- rungs- maß- nah- men entl…		
2	54.17	13.03	Ros- tock, Han- se- und Uni- ver- si- täts- stadt	Ros- tock, Han- se- und Uni- ver- si- täts- stadt	13003	13003000	Am Markt	6	Was- ser- lei- tung	NaN	2024-03-28 07:00:00	2024-03-28 18:00:00	Ein- rich- tung Trink- was- ser- schie- ber der Stra- ße, Si- cher…	22.458333
3	54.08	12.13	Ros- tock	Ros- tock, Han- se- und Uni- ver- si- täts- stadt	13061	13061021	Kopf- stock- str.	5	Ge- bäu- de- sa- nie- rung	NaN	2024-01-30 00:00:00	NaN	NaN	82.000000

	latitude	longitude	[overlapping headers]						schluessel	sparte	von	nach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
4	54.07[illegible]	12.13[illegible]	[illegible]-a6bd-4c[illegible]-b25b-09fe10a23a2[illegible]	Rostock	13003[illegible]	Rostock, Hansestadt und Universitätsstadt	13003000[illegible]	Rostock, Hansestadt und Universitätsstadt	130030000[illegible]	Ziolkowskistr.	[illegible]0192[illegible]	Wasserleitung	NaN	NaN	2023-03-06 00:00:00+01:00	2024-08-01 00:00:00+02:00	Sicherungsmaßnahmen entlang der Straße, Sicher...	Sanierung Mischwasser- und Trinkwasserleitung	513.958333

```
baust.tail(3)
```

	latitude	longitude	[overlapping headers]						schluessel	sparte	von	nach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
201	54.08[illegible]	12.1[illegible]	[illegible]-b521-b6[illegible]91455	Rostock	1300[illegible]	Rostock, Hanse- und Univer- si-	13003000[illegible]	Rostock, Hanse- und Univer- si-	130030000[illegible]	Südring	[illegible]07400	Gleisbau	Platz der Freundschaft (07650)	Goetheplatz (02960)	2022-05-[illegible] 00:00:00+0[illegible]	2025-[illegible]-01 00:00:00+01:00	Sicherungsmaßnahmen entlang	Baustelleneinrichtung / Belie-	973.041667

	latitude	longitude	[illegible]	[illegible]	[illegible]	[illegible]	[illegible] schluessel	sparte	von	nach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
					täts-stadt	täts-stadt							der Straße, Sicher.	fe-rung für Neubau…	
202	54.151[illegible]	[illegible]	[illegible]	[illegible]tock	[illegible]tock, Hanse- und Universitätsstadt	[illegible]tock, Hanse- und Universitätsstadt	[illegible]lerring	Hochbau	45	NaN	2023-[illegible] 00:00[illegible]	2025-[illegible]	[illegible]rungsmaßnahmen entlang der Straße, Sicher…	Bau-[illegible]ße für Neubau Hort Taklerring	822.000000
203	54.08[illegible]	[illegible]	[illegible]ae98-b0bbd9703f18	[illegible]tock	[illegible]tock, Hanse- und Universitätsstadt	[illegible]tock, Hanse- und Universitätsstadt	[illegible]hardingstr.	Telefonnetz	18	29	2024-[illegible] 07:00[illegible]	2024-[illegible]	[illegible]rungsmaßnahmen entlang der Straße, Sicher…	Breitbandausbau	3.458333

```python
baust.sample(4)
```

	latitude	longitude	[illegible]	[illegible]	[illegible]	strasse_schluessel	sparte	von	nach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
116	[illegible]	[illegible]	[illegible] d2a2[illegible]709b	[illegible] tock, Hanse- und Universitätsstadt	[illegible] tock, Hanse- und Universitätsstadt	[illegible] Strom	Wasserleitung	119	122	2024-[illegible] 07:00[illegible]	[illegible]	[illegible] rungsmaßnahmen entlang des Gehwegs	[illegible] lung Hausanschluss	39.416667
65	[illegible]	[illegible]	[illegible] tock	[illegible] tock, Hanse- und Universitätsstadt	[illegible] tock, Hanse- und Universitätsstadt	[illegible] der Freundschaft	Wasserleitung	NaN	NaN	2024-[illegible] 07:00[illegible]	[illegible]	[illegible] rungsmaßnahmen entlang der Straße, Vollsp...	[illegible] reparaturen	10.458333
16	[illegible]	[illegible]	[illegible] tock	[illegible] tock, Hanse- und Universitätsstadt	[illegible] tock, Hanse- und Univer-	[illegible] Schlesinger-Str.	Kabelnetz	51	NaN	2024-[illegible] 00:00[illegible]	[illegible] 01:00	[illegible] rungsmaßnahmen ent-	NaN	9.000000

	latitude	longitude	uuid	kreis_name	kreis_schluessel	gemeindeverband_name	gemeindeverband_schluessel	gemeinde_name	gemeinde_schluessel	strasse_name	strasse_schluessel	sparte	von	nach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
				sitätsstadt		sitätsstadt											lang der Straße, Sicher...		
136	54.06[illegible]	12.1[illegible]	[illegible]-b6e1-4ec2-81db-5d81a1477deb	Rostock, Hanse- und Universitätsstadt	13003	Rostock, Hanse- und Universitätsstadt	13003000	[illegible]	130030000	Dalwitzhof	01750	Stromnetz	5	7	2024-[illegible] 00:00:00+00:00	2024-05-01 00:00:00+00:00	Sicherungsmaßnahmen entlang der Straße, Sicher...	Verlegung Erdkabel - Hausanschluss	57.958333

Diese Funktionen zeigt uns zwar im ersten Eindruck viele Spalten, aber nicht immer alle, wenn der Bildschirmplatz nicht ausreicht. Deshalb ist es bei solchen Datensätzen mit vielen Spalten sinnvoll alle Spaltennamen in einem Datensatz aufzulisten.

```
baust.columns
```

```
Index(['latitude', 'longitude', 'uuid', 'kreis_name', 'kreis_schluessel',
       'gemeindeverband_name', 'gemeindeverband_schluessel', 'gemeinde_name',
       'gemeinde_schluessel', 'strasse_name', 'strasse_schluessel', 'sparte',
       'von', 'nach', 'baubeginn', 'bauende', 'verkehrsbeeintraechtigungen',
       'baumassnahme', 'dauer'],
      dtype='str')
```

Als nächstes überprüfen wir die Datentypen der Daten. Obwohl einige, z.B. `kreis_name` und `baumassnahme` aussehen wie ein String, könnten sie tatsächlich einen anderen Typ haben. Dies kann über das Attribut `.dtypes` abgefragt werden:

```
baust.dtypes
```

```
latitude                      float64
longitude                     float64
uuid                              str
kreis_name                        str
kreis_schluessel              float64
gemeindeverband_name              str
gemeindeverband_schluessel    float64
gemeinde_name                     str
gemeinde_schluessel           float64
strasse_name                      str
strasse_schluessel                str
sparte                            str
von                               str
nach                              str
baubeginn                         str
bauende                           str
verkehrsbeeintraechtigungen       str
baumassnahme                      str
dauer                         float64
dtype: object
```

Die Datentypen geben an, dass sowohl `kreis_name` und `baumassnahme` keine Strings sondern Objekte sind. Hier bedeutet es Zeichenfolgen, im Prinzip können es auch komplexere Objekte sein. Wenn wir zum Beispiel mal die Anzahl der Werte für `kreis_name` berechnen

```
baust.kreis_name.count()
```

```
np.int64(203)
```

so ist diese Zahl niedriger als die Anzahl der Zeilen die `baust.shape` zurückgeliefert hat. Das liegt an fehlenden Werten. Dazu mehr im Abschnitt Datenvorverarbeitung.

Was sind die Werte?

Einer der ersten Schritte mit einem neuen Datensatz ist es, die Werte der relevanten Variablen zu überprüfen. Die Pandas-Bibliothek enthält viele Tools für die beschreibenden Datenanalyse. Bei den kategorischen Variablen möchten wir normalerweise die expliziten Werte sehen, bei den numerischen können wir die minimalen und maximalen Werte überprüfen.

Verschiedene diskrete Werte

Was sind die möglichen Werte für *Kreisname*? Ein schneller Blick auf die Daten zeigt, dass es einige Werte für die Spalte `sparte`, aber gibt es noch mehr? Das können wir mit der Methode `.unique()` abfragen:

```
baust.sparte.unique()
```

```
<ArrowStringArray>
[  'Fernwärmeleitung',         'Grünpflege',      'Wasserleitung',
   'Gebäudesanierung',            'Hochbau',         'Straßenbau',
          'Stromnetz',          'Kabelnetz',         'Gasleitung',
        'Telefonnetz',  'Lichtsignalanlage', 'Straßenbeleuchtung',
            'privat',           'Gleisbau',       'Kranarbeiten']
Length: 15, dtype: str
```

Das Ergebnis zeigt uns, welche unterschiedlichen Sparten in den Baustellendaten vorkommen, zum Beispiel `Hochbau`, `Straßenbau` oder `Gebäudesanierung`. Wir können auch ein technisches Detail erkennen, nämlich dass das Ergebnis als ein numpy-Array zurückgegeben wird.

Tip

Ein Wort über Methoden und Methodenverkettung. Lassen Sie uns den vorherigen Befehl, `baust.sparte.unique()`, überprüfen. Dieser enthält drei Komponenten, und der Prozess besteht darin, diese in einer sequenziellen Reihenfolge anzuwenden:

- `baust` ist der Datensatz. Das ist das, womit wir anfangen.
- `.sparte` ist ein *Attribut* von `baust`. Hier bedeutet es, das die Spalte `sparte` aus dem Datensatz extrahiert wird und als Serie zurückgegeben wird.
- Schließlich ist `.unique()` eine *Methode*, welche die Serie an Werten auf ihrer linken Seite analysiert (hier die Serie `sparte`), und diese analysiert um die eindeutigen Werten zu identifizieren.

Einige der Methoden können entweder auf den gesamten Datenrahmen oder auf einzelne Variablen angewendet werden. Wenn sie auf den gesamten Datenrahmen angewendet werden, gelten sie für jede einzelne Variable separat, sozusagen wie eine Schleife über einzelne Variablen, auf die sie angewendet werden.

Häufig ist es nicht nur interessant bei kategorischen Werten zu verstehen, welche eindeutigen Werte auftreten, sondern wie oft diese Auftreten. Hierfür nutzt man die Funktion

```
baust.sparte.value_counts()
```

```
sparte
Hochbau               40
Gebäudesanierung      35
Straßenbau            35
Wasserleitung         25
```

```
Fernwärmeleitung      18
Telefonnetz           16
Kabelnetz             11
Stromnetz              8
Grünpflege             4
Lichtsignalanlage      3
Straßenbeleuchtung     3
Gleisbau               2
Kranarbeiten           2
Gasleitung             1
privat                 1
Name: count, dtype: int64
```

Das gibt uns schon mal ein gutes Bild der Sparten. Aber wie sieht es mit der Spalte `baumassnahme` aus?

```
baust.baumassnahme.unique()
```

```
<ArrowStringArray>
[                                              'FW-Leitung i.a. der
Hanseatic',

'Baumpflanzung',
                                                  'Einbau
Trinkwasserschieber',

nan,
                                 'Sanierung Mischwasser- und
Trinkwasserleitung',

'Containerstellung',
                                             'Kennzeichnung
Baustellenausfahrten',
 'Wartungs- und Instandsetzungsarbeiten innerhalb der konzessionierten
Strecke',
                                      'Grünraumarbeiten, Sanierung
Außenanlagen',
                                        'Ausschleifung - Muffung, StS
Bussebart',
 ...
                                                 'Sanierung
Trinkwasserleitung',
                                                        'Grundhafte
Erneuerung',
                                                           'Neubau
Ampelanlage',
                                                        'Erneuerung TW-
```

```
Leitung',
                                                    'Anschluss
Personenbahnhof',
                              'Arbeiten Hausanschluss Trink- und
Schmutzwasser',
                     'Errichtung einer Kabelbrücke für E-Versorgung
Baustelle',
                                        'Anbindung Breitband Stadtentsorgung
HRO',
             'Baustelleneinrichtung / Belieferung für Neubau
Goetheplatzbrücke',
                                           'Baustraße für Neubau Hort
Taklerring']
Length: 141, dtype: str
```

Diese Liste gibt ein viel komplexeres Bild und listet eine Vielzahl an Typen von Baumaßnahmen. Wenn wir hier die Funktion `value_counts()` verwenden, so listet diese auch nicht alle Werte auf:

```
baust.baumassnahme.value_counts()
```

```
baumassnahme
Breitbandausbau                                                    12
Erneuerung Straßenbeleuchtung                                       3
Sanierung Trinkwasserleitung                                        3
Baumpflanzung                                                       2
Sanierungsarbeiten                                                  2
                                                                   ..
Arbeiten Hausanschluss Trink- und Schmutzwasser                     1
Errichtung einer Kabelbrücke für E-Versorgung Baustelle             1
Anbindung Breitband Stadtentsorgung HRO                             1
Baustelleneinrichtung / Belieferung für Neubau Goetheplatzbrücke    1
Baustraße für Neubau Hort Taklerring                                1
Name: count, Length: 140, dtype: int64
```

Bei Datensätzen mit hunderttausenden von Zeilen kann das sehr lange dauern und kein verwertbares Ergebnis liefern. Deshalb ist es meist sinnvoll erstmal zu prüfen wie viele eindeutige Werte existieren. Diese Anzahl können wir mit der Methode `.nunique` abfragen:

```
baust.baumassnahme.nunique()
```

```
140
```

Numerische Werte

Für numerische Variablen sollte es vermieden werden `unique()` weil diese meist sehr große Ergebnismengen zurück geben, die selten nützlich sind. Stattdessen ist interessante den Werte-

bereich der Variable zu verstehen (siehe „Normierung“). Dies gibt einen schnellen Überblick darüber, ob die Werte plausibel sind, z.B. ob negative Alterswerte auftreten oder ungewöhnlich große Alterswerte:

```
baust.longitude.min(), baust.longitude.max()
```

```
(np.float64(12.0120276853359), np.float64(12.2248919062465))
```

```
baust.latitude.min(), baust.latitude.max()
```

```
(np.float64(54.057580297901), np.float64(54.2396815337323))
```

Der Wertebereich für beide ist für Rostock durchaus plausibel und wir machen uns daher keine Sorgen über zu große oder zu kleine Werte. Siehe Abschnitt „Entfernen unerwünschter Variablen“ unten, um fehlende Werte zu erkennen und zu analysieren.

Pandas bietet die Methode `describe()` an, die es erlaubt diese Statistiken für alle numerische Spalten in einem Aufruf zu extrahieren. Als Ergebnis erhalten wir einen neun Dataframe mit den Statistiken.

```
baust.describe()
```

	latitude	longitude	kreis_schluessel	gemeindeverband_schluessel	gemeinde_schluessel	dauer
count	204.000000	204.000000	203.0	203.0	2.030000e+02	204.000000
mean	54.104472	12.110911	13003.0	130030000.0	1.300300e+11	229.857230
std	0.031356	0.034811	0.0	0.0	0.000000e+00	278.425382
min	54.057580	12.012028	13003.0	130030000.0	1.300300e+11	0.166667
25%	54.083985	12.084762	13003.0	130030000.0	1.300300e+11	30.572917
50%	54.090294	12.116659	13003.0	130030000.0	1.300300e+11	97.125000
75%	54.124206	12.136534	13003.0	130030000.0	1.300300e+11	336.750000
max	54.239682	12.224892	13003.0	130030000.0	1.300300e+11	1347.000000

Zählungen und Anteile

Eine weitere häufige Aufgabe, mit der wir konfrontiert sind, besteht darin, Werte zu zählen oder Anteile von bestimmten Werten zu berechnen. Die Methode `.value_counts` (siehe oben) kann verwendet werden, um die Anzahl kontinuierlicher Werte zu erhalten. Bei numerischen Werten funktioniert das aufgrund der Vielfalt and Werte nicht gut. Außerdem hilft es auch nicht Fragen die von Bedingungen abhängig sind. Zum Beispiel wollen wir wissen wie viele Baustellen im

Jahr 2024 begonnen worden sind. Dafür müssen wir einen logischen Vektor erstellen, der die Bedingung beschreibt, und diesen summieren.

```
seit2024 = baust.baubeginn > "2024-01-01 00:00:00" # Erzeuge eine logische
Variable die True ist wenn der baubeginn nach 2024 ist
seit2024.sum() # Zähle die Baustellen
```

```
np.int64(127)
```

Lassen Sie uns diese Schritte genauer erklären. Zuerst die Zeile

```
seit2024 = baust.baubeginn > "2024-01-01 00:00:00"
seit2024.head(5)
```

```
0     True
1     True
2     True
3     True
4    False
Name: baubeginn, dtype: bool
```

Erzeuge eine logische Variable die `True` oder `False` ist, wenn das Datum des Baubeginns nach 2024 liegt (um genau zu sein machen wir hier einen Stringvergleich und keinen Datumsvergleich. Da wir die Spalte nicht in ein Datum umgewandelt haben.). Ein Beispiel für diese Variable sieht so aus:

Als nächstes addiert `seit2024.sum()` diese Werte zusammen - denken Sie daran, sobald wir mit logischen Werten rechnen, wird `True` in „1“ und `False` in „0“ umgewandelt, und letzteres spielt beim Addieren keine Rolle. Es ist also so, als würden wir Baustellen zählen. In der Praxis müssen Sie keine separate Variable erstellen, sondern können es direkt berechnen

```
(baust.baubeginn > "2024-01-01 00:00:00").sum()
```

```
np.int64(127)
```

Die Berechnung von Proportionen kann auf ähnliche Weise erfolgen. Statt `.sum` verwenden wir einfach `.mean()`:

```
(baust.baubeginn > "2024-01-01 00:00:00").mean()
```

```
np.float64(0.6225490196078431)
```

Also gibt es 62% der Baustellen seit 2024.

Filtern von Daten

Schließlich, lassen wollen wir die Teilmengen der Baustellen erhalten, die entweder im „Hochbau“ oder „Straßenbau“ liegen. Dies kann mit der Methode `.isin()` durchgeführt werden, um zu überprüfen, ob ein String in einer gegebenen Liste vorhanden ist:

```
ab = baust[baust.sparte.isin(["Hochbau", "Straßenbau"])]
ab.head(5)
```

	latitude	longitude	[illegible]	sparte	von	nach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
5	54.14[illegible]	12.0665[illegible]	[illegible] Rostock, Hanse- und Universitätsstadt | Rostock, Hanse- und Universitätsstadt | Blankenburger Str. [illegible]	Hochbau	25	26	2023-[illegible] 07:00[illegible]	[illegible]	[illegible] rungsmaßnahmen entlang der Straße, Verkeh…	[illegible] nerstellung	[illegible] 132.375000
8	54.14[illegible]	12.0[illegible]	[illegible] Rostock, Hanse- und Universitätsstadt | Rostock, Hanse- und Universitätsstadt | Möllner Str. [illegible]	Hochbau	NaN	NaN	2024-[illegible] 07:00[illegible]	[illegible]	[illegible] rungsmaßnahmen entlang der Straße	Kennzeichnung Baustellenausfahrten	646.458333

	latitude	longitude	[illegible: overlapping headers]						sparte	von	nach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
9	54.135[illegible]	12.0913463	Red8-[illegible]-a7b1-4bf1-8e3f-95c569f480c6	Rostock	13003	Rostock, Hanse- und Universitätsstadt	13003000	Schmarl Dorf	0180	Straßenbau	NaN	NaN	2024-01-01 00:00:00+00:00	2025-01-01 00:00:00+00:00	Sicherungsmaßnahmen entlang der Straße, Verkeh…	Wartungs- und Instandsetzungsarbeiten innerhal…	366.000000
13	54.179[illegible]	12.0718[illegible]	Rd03-7a03-4637-[illegible]-f7519b29a64f	Rostock	13003	Rostock, Hanse- und Universitätsstadt	13003000	Budenessenstr.	0476	Hochbau	5	NaN	2024-03-22 07:00:00+00:00	2024-03-28 18:00:00+00:00	Sicherungsmaßnahmen entlang der Straße	Sanierungsarbeiten	6.458333
14	54.089[illegible]	12.1038[illegible]	R275-[illegible]-aebe-497f-b981-1e8f09b81473	Rostock	13003	Rostock, Hanse- und Universitätsstadt	13003000	Waldemarstr.	0941	Straßenbau	20	34	2024-03-04 07:00:00+00:00	2024-03-28 18:00:00+00:00	Halbseitige Sperrung, Sicherungs-	Schutzrohrverlegung	24.458333

lagetitude	longitude	kreis_name	kreis_schluessel	gemeindeverband_name	gemeindeverband_schluessel	gemeinde_name	gemeinde_schluessel	strasse_name	strasse_schluessel	sparte	vonnach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer

Auswahl von Variablen

Häufig enthalten Datensätze Variablen, die für die aktuelle Analyse irrelevant sind. In einem solchen Fall möchten wir entweder nur die relevanten auswählen (wenn es nur wenige solcher Variablen gibt) oder die irrelevanten entfernen (wenn es nicht zu viele davon gibt). Das Entfernen irrelevanter Informationen hilft uns, die Daten besser zu sehen, sie besser zu verstehen und somit weniger Codierfehler zu haben. Es hilft auch bei bestimmten Transformationen, bei denen wir jetzt möglicherweise auf den gesamten reduzierten Datenrahmen anstatt nur auf ausgewählte Variablen arbeiten möchten. Schließlich hilft es auch, Speicher zu sparen und die Verarbeitungsgeschwindigkeit zu erhöhen.

Auswahl der gewünschten Variablen

Wir haben besprochen, wie man die gewünschten Variablen auswählt, aber wir werden es hier auch kurz überprüfen. Sie können nur die gewünschten Variablen mit `dataframe[["var1", "var2", ..."]]` auswählen.

Der Baustellendatensatz enthält zum Beispiel viele Spalten wie „uuid“, „kreis_name“, „kreis_schluessel“, „gemeindeverband_name“, „gemeindeverband_schluessel“, „gemeinde_name“, „gemeinde_schluessel“, „strasse_name“, oder „strasse_schluessel“ die redundant sind und nicht gebraucht werden. Deshalb können wir den Datensatz in seiner Komplexität gut reduzieren, indem wir einfach nur die relevanten Spalten auswählen:

```
baust[["sparte", "baumassnahme", "verkehrsbeeintraechtigungen", "baubeginn",
"bauende", "latitude", "longitude", "dauer"]].sample(4)
```

	sparte	baumassnahme	verkehrsbeeintraechtigungen	baubeginn	bauende	latitude	longitude	dauer
156	Wasserleitung	Trennung Trinkwassergrundstücksanschluss	Sicherungsmaßnahmen entlang der Straße, Sicher…	2024-03-25 00:00:00+01:00	2024-03-28 00:00:00+01:00	54.068818	12.073450	3.000000
29	Gebäudesanierung	Sanierung Altbau	Sicherungsmaßnahmen entlang der Straße, Sicher…	2023-08-31 00:00:00+02:00	2024-09-28 00:00:00+02:00	54.086933	12.109909	394.000000
55	Gebäudesanierung	NaN	NaN	2024-03-15 00:00:00+01:00	2024-04-13 00:00:00+02:00	54.090554	12.134640	28.958333
123	Stromnetz	Verlegung Erdkabel	Sicherungsmaßnahmen entlang des Gehwegs, Sperr…	2024-04-02 07:00:00+02:00	2024-04-12 18:00:00+02:00	54.115538	12.059869	10.458333

Bitte beachten Sie, dass die Spalten in der Reihenfolge zurückgegeben werden, in der sie in der Auswahl aufgeführt sind, nicht in der Originalreihenfolge.

Anstatt den gesamten Datensatz zu laden und später die gewünschten Variablen auszuwählen, können wir auch angeben, welche Spalten mit `pd.read_csv` gelesen werden sollen.

```
pd.read_csv("../data/Baustellen/baustellen_flat.csv",
            usecols=["latitude", "longitude", "sparte", "baubeginn",
"bauende", "verkehrsbeeintraechtigungen", "baumassnahme", "dauer"]).sample(4)
```

	latitude	longitude	sparte	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
56	54.097021	12.093661	Gebäudesanierung	2024-01-22 00:00:00+01:00	2024-04-01 00:00:00+02:00	NaN	NaN	69.958333
70	54.071622	12.120165	Stromnetz	2024-03-18 07:00:00+01:00	2024-03-28 18:00:00+01:00	Sicherungsmaßnahmen entlang des Gehwegs	Herstellung Hausanschluss	10.458333
188	54.143706	12.068219	Hochbau	2022-03-07 09:00:00+01:00	2024-05-31 18:00:00+02:00	Sicherungsmaßnahmen entlang der Straße, Sicher…	Neubau Mehrfamilienhaus mit Tiefgarage	816.333333
95	54.064148	12.105620	Wasserleitung	2023-06-12 00:00:00+02:00	2024-05-04 00:00:00+02:00	Sicherungsmaßnahmen entlang der Straße, Sicher…	Sanierung Trinkwasserleitung 3. BA	327.000000

Dies erreicht ein ähnliches Ergebnis, obwohl die Spalten jetzt in der ursprünglichen Reihenfolge und nicht in der angegebenen Reihenfolge sind.

Entfernen unerwünschter Variablen

Alternativ können wir, wenn wir die meisten Variablen behalten möchten, stattdessen angeben, welche entfernt werden sollen. Dies kann mit der Methode `.drop` erfolgen. Im Folgenden lassen wir eine große Anzahl von Variablen fallen:

```
baust.drop(["uuid", "kreis_name", "kreis_schluessel", "gemeindeverband_name",
"gemeindeverband_schluessel", "gemeinde_name", "gemeinde_schluessel",
"strasse_name", "strasse_schluessel"],
            axis=1 ).sample(4)
```

	latitude	longitude	sparte	von	nach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
88	54.083090	12.108242	Gebäudesanierung	15	NaN	2024-04-22 00:00:00+02:00	2024-06-08 00:00:00+02:00	NaN	NaN	47.000000
119	54.174596	12.092916	Kabelnetz	3	6	2024-02-05 07:00:00+01:00	2024-03-28 18:00:00+01:00	NaN	Kabelsanierung	52.458333
113	54.153554	12.069177	Straßenbau	NaN	NaN	2024-01-04 07:00:00+01:00	2024-03-28 17:00:00+01:00	halbseitige Sperrung, Sicherungsmaßnahmen entl…	Rohrleitungsbau	84.416667
183	54.082927	12.087269	Straßenbau	23	NaN	2024-04-02 00:00:00+02:00	2024-04-11 02:00:00+02:00	halbseitige Sperrung, Sicherungsmaßnah-	Reparatur Gehweg	9.000000

latitude	longitude	sparte	von	nach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
							men entl…		

Beachten Sie, dass wir `.drop` mitteilen müssen, dass wir Spalten mit diesen Namen zu entfernen und nicht Zeilen mit diesem Index. Dies ist, was das Argument `axis=1` bewirkt. Andernfalls erhalten wir einen Fehler:

```
baust.drop(["uuid", "kreis_name", "kreis_schluessel", "gemeindeverband_name",
"gemeindeverband_schluessel", "gemeinde_name", "gemeinde_schluessel",
"strasse_name", "strasse_schluessel"]).sample(4)
```

```
KeyError: "['uuid', 'kreis_name', 'kreis_schluessel', 'gemeindeverband_name',
'gemeindeverband_schluessel', 'gemeinde_name', 'gemeinde_schluessel',
'strasse_name', 'strasse_schluessel'] not found in axis"
[31m---------------------------------------------------------------------------[
[39m
[31mKeyError[39m                                  Traceback (most recent
call last)
[36mCell[39m[36m [39m[32mIn[25][39m[32m, line 1[39m
[32m----> [39m[32m1[39m baust.drop([[33m"uuid"[39m,
[33m"kreis_name"[39m, [33m"kreis_schluessel"[39m,
[33m"gemeindeverband_name"[39m, [33m"gemeindeverband_schluessel"[39m,
[33m"gemeinde_name"[39m, [33m"gemeinde_schluessel"[39m,
[33m"strasse_name"[39m,
[33m"strasse_schluessel"[39m]).sample([32m4[39m)

[36mFile [39m[32m~/Documents/code/Lehre/
IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/
frame.py:6288[39m, in [36mDataFrame.drop[39m[34m(self, labels, axis,
index, columns, level, inplace, errors)[39m
[32m   6284[39m                 weight  [32m250.0[39m
[32m150.0[39m
[32m   6285[39m         falcon  speed   [32m320.0[39m
[32m250.0[39m
[32m   6286[39m                 weight  [32m1.0[39m     [32m0.8[39m
[32m   6287[39m         """
[32m-> [39m[32m6288[39m         return super().drop(
```

```
[32m   6289[39m             labels=labels,
[32m   6290[39m             axis=axis,
[32m   6291[39m             index=index,

[36mFile [39m[32m~/Documents/code/Lehre/
IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/
generic.py:4644[39m, in [36mNDFrame.drop[39m[34m(self, labels, axis,
index, columns, level, inplace, errors)[39m
[32m   4640[39m         obj = self
[32m   4641[39m
[32m   4642[39m         [38;5;28;01mfor[39;00m axis, labels
[38;5;28;01min[39;00m axes.items():
[32m   4643[39m             [38;5;28;01mif[39;00m labels
[38;5;28;01mis[39;00m [38;5;28;01mnot[39;00m
[38;5;28;01mNone[39;00m:
[32m-> [39m[32m4644[39m                 obj = obj._drop_axis(labels,
axis, level=level, errors=errors)
[32m   4645[39m
[32m   4646[39m         [38;5;28;01mif[39;00m inplace:
[32m   4647[39m             self._update_inplace(obj)

[36mFile [39m[32m~/Documents/code/Lehre/
IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/
generic.py:4686[39m, in [36mNDFrame._drop_axis[39m[34m(self, labels,
axis, level, errors, only_slice)[39m
[32m   4682[39m                 [38;5;28;01mif[39;00m
[38;5;28;01mnot[39;00m isinstance(axis, MultiIndex):
[32m   4683[39m                     [38;5;28;01mraise[39;00m
AssertionError([33m"axis must be a MultiIndex"[39m)
[32m   4684[39m                 new_axis = axis.drop(labels, level=level,
errors=errors)
[32m   4685[39m             [38;5;28;01melse[39;00m:
[32m-> [39m[32m4686[39m                 new_axis = axis.drop(labels,
errors=errors)
[32m   4687[39m             indexer = axis.get_indexer(new_axis)
[32m   4688[39m
[32m   4689[39m         [38;5;66;03m# Case for non-unique axis[39;00m

[36mFile [39m[32m~/Documents/code/Lehre/
IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/
indexes/base.py:7268[39m, in [36mIndex.drop[39m[34m(self, labels,
errors)[39m
[32m   7266[39m [38;5;28;01mif[39;00m mask.any():
```

```
[32m   7267[39m     [38;5;28;01mif[39;00m errors !=
[33m"[39m[33mignore[39m[33m"[39m:
[32m-> [39m[32m7268[39m         [38;5;28;01mraise[39;00m
[38;5;167;01mKeyError[39;00m([33mf[39m[33m"[39m[38;5;132;01m{[39;00mlabels[mask].to
[38;5;132;01m}[39;00m[33m not found in axis[39m[33m"[39m)
[32m   7269[39m     indexer = indexer[~mask]
[32m   7270[39m [38;5;28;01mreturn[39;00m
[38;5;28mself[39m.delete(indexer)

[31mKeyError[39m: "['uuid', 'kreis_name', 'kreis_schluessel',
'gemeindeverband_name', 'gemeindeverband_schluessel', 'gemeinde_name',
'gemeinde_schluessel', 'strasse_name', 'strasse_schluessel'] not found in
axis"
```

Ohne `axis=1` interpretiert Pandas die übergebenen Namen als **Zeilenindex** und nicht als Spaltennamen. Da diese Bezeichner nicht im Zeilenindex vorkommen, entsteht der gezeigte `KeyError`.

Insbesondere bedeutet dies, dass `.drop` einen Fehler verursacht, wenn wir eine nicht vorhandene Spalte entfernen möchten. Dies kann Probleme verursachen, wenn Daten in Notebooks überschrieben werden: Im ersten Durchlauf entfernen Sie eine Variable, und wenn Sie dieselbe Zelle erneut ausführen, ist die Variable weg und Sie erhalten einen Fehler. Wie immer sollten Sie entweder Daten umbenennen, das erneute Ausführen des Datenbereinigungsschritts vermeiden oder Fehlerbehandlung einbeziehen.

Gruppierte Operationen

Wie andere große DataFrame-Bibliotheken unterstützt auch Pandas gruppierte Operationen. Gruppierte Operationen werden im Wesentlichen wie folgt durchgeführt: Zuerst wird die Daten anhand der Gruppierungsvariablenwerte in Gruppen aufgeteilt. Zweitens werden die folgenden Operationen auf allen Gruppen unabhängig voneinander durchgeführt. Und schließlich werden alle Ergebnisse wieder zusammengeführt, zusammen mit Gruppenindikatoren.

Gruppieren in Pandas kann mit der Methode `.groupby` durchgeführt werden. Es erwartet eine Liste von Gruppierungsvariablen. Wir demonstrieren dies, indem wir den Baustellendauer nach Sparte berechnen. `.groupby` erstellt lediglich ein gruppiertes Datenframe, der dann um eine Aggregationsfunktion erweitert wird um einen Ergebnisdataframe zu erhalten.

```
baust.groupby("sparte").dauer.mean()
```

```
sparte
Fernwärmeleitung       75.138889
Gasleitung            185.750000
Gebäudesanierung      179.201786
Gleisbau              939.541667
Grünpflege             68.593750
```

```
Hochbau               466.585417
Kabelnetz              51.462121
Kranarbeiten            3.500000
Lichtsignalanlage     260.472222
Straßenbau            295.767857
Straßenbeleuchtung     46.416667
Stromnetz              71.640625
Telefonnetz            54.661458
Wasserleitung         198.724167
privat                  2.250000
Name: dauer, dtype: float64
```

Das Ergebnis ist eine Serie mit zwei Werten (die numerische *dauer*) mit dem Index der kategorischen Werte von *sparte*.

Wir können auch nach mehr als einer Variable gruppieren. In diesem Fall müssen wir diese als Liste angeben, z.B. um die Baustellendauer nach Sparte und Baumaßnahme zu berechnen, können wir das tun.

```
baust.groupby(["sparte", "baumassnahme"]).dauer.mean()
```

```
sparte            baumassnahme
Fernwärmeleitung  Erweiterung Fernwärme Warnemünde
316.375000
                  FW-Leitung i.a. der Hanseatic
39.375000
                  Fernwärme
39.333333
                  Fernwärmeerweiterung
186.041667
                  Fernwärmehausanschluss
32.000000
                                                                    ...
Wasserleitung     Sanierung Wasser/ Abwasser in versch. Bauphasen
494.958333
                  Schachtdeckel Havarie Übernahme VRAO Az.
190.000000
                  Trennung Trinkwassergrundstücksanschluss
3.000000
                  Umverlegung Trinkwasserleitung, SW & RW-Kanal
303.000000
privat            Fundamentarbeiten Balkonanlage -Betonpumpe & Betonmischer-
2.250000
Name: dauer, Length: 145, dtype: float64
```

Die Methode `.groupby` unterstützt weitere Optionen, z.B. kann man die Gruppenindikatoren als Variablen behalten anstatt als Index.

Zeichenkettenoperationen

Pandas öffnet Zeichenkettenvariablen für eine große Liste von Zeichenkettenfunktionen mit dem Attribut `.str`. Diese replizieren größtenteils das *re* Modul, aber die Syntax ist anders und oft sind auch die Funktionsnamen unterschiedlich. Wir gehen hier durch eine Reihe von Beispielen, nämlich

- `str.contains` zum Suchen von Mustern in Zeichenketten
- `str.match` um festzustellen, ob eine Zeichenkette mit einem bestimmten Muster beginnt
- `str.replace` um Teile von Zeichenketten zu ersetzen
- `str.split` zum Zerteilen von Zeichenketten in Wörter

Viele der Zeichenkettenfunktionen nutzen reguläre Ausdrücke, daher ist es nützlich, eine Vorstellung von den Grundlagen regulärer Ausdrücke zu haben.

Lassen Sie uns Baustellendaten verwenden und analysieren, ob es ein regelmäßiges Muster in den Baumaßnahmen gibt. Wenn wir nur `unique` aufrufen, erhalten wir eine sehr unübersichtliche Liste.

```
baust.baumassnahme.unique()[:10]
```

```
<ArrowStringArray>
[                                                'FW-Leitung i.a. der
Hanseatic',

'Baumpflanzung',
                                                    'Einbau
Trinkwasserschieber',

nan,
                                 'Sanierung Mischwasser- und
Trinkwasserleitung',

'Containerstellung',
                                             'Kennzeichnung
Baustellenausfahrten',
 'Wartungs- und Instandsetzungsarbeiten innerhalb der konzessionierten
Strecke',
                                      'Grünraumarbeiten, Sanierung
Außenanlagen',
                                        'Ausschleifung - Muffung, StS
Bussebart']
Length: 10, dtype: str
```

Allerdings sehen wir auch, dass einige Wörter wie z.B. „Sanierung“ wiederholt vorkommen. Vieleicht ist es sinnvoller, die Zeichenkette in einzelne Worte zu zerlegen und diese zu zählen. Hierfür müssen wir mehrere Methoden verknüpfen, indem wir die Zeichenkette zuerst in Listen an Wörten mit `split` an allen vorkommenden Leerzeichen und Satzzeichen zerlegen. Mit `stack()`

kombinieren wir diese Liste in eine neue Serie und zählen dann mit `value_counts()` wie häufig welche Wörter vorkommen.

```
baust.baumassnahme.str.split(expand=True).stack().value_counts()
```

```
Neubau             25
und                21
Sanierung          19
Breitbandausbau    13
für                12
                   ..
HRO                 1
Belieferung         1
Baustraße           1
Hort                1
Taklerring          1
Name: count, Length: 283, dtype: int64
```

Siehe da: Worte wie „Neubau“ und „Sanierung“ kommen durchaus oft vor.

Finde Muster in Zeichenketten

Als ersten Schritt wollen wir alle Baumaßnamen identifizieren, die das Wort „Sanierung“ enthalten. Muster in Zeichenfolgen können mit `str.contains(Muster)` identifiziert werden. Diese Funktion gibt einen Vektor von Wahrheitswerten zurück, je nachdem, ob der ursprüngliche Zeichenvektor das Muster enthält:

```
baust.sparte.str.contains("sanierung", na=False, regex=False,
case=False).head(4)
```

```
0    False
1    False
2    False
3     True
Name: sparte, dtype: bool
```

Das Argument `na` gibt an, was mit fehlenden Werten zu tun ist, hier zwingen wir sie, den Wert `false` zurückzugeben. `regex=False` bedeutet, dass das Muster, das wir angeben, kein regulärer Ausdruck ist. Einfache Muster sind einfacher und schneller zu verwenden als reguläre Ausdrücke, aber letztere sind viel leistungsfähiger, aber auch deutlich komplizierter. Der Parameter `case` gibt an, dass die Funktion nicht auf Groß/Kleinschreibung achten soll.

Jetzt wollen wir uns mal die Baumaßnahmen ansehen, die den Begriff Sanierung enthalten

```
baust.baumassnahme[baust.baumassnahme.str.contains("sanierung", na=False,
regex=False, case=False)].value_counts()
```

```
baumassnahme
Sanierung Trinkwasserleitung                                          3
Sanierungsarbeiten                                                    2
Sanierung Mischwasser- und Trinkwasserleitung                         1
Grünraumarbeiten, Sanierung Außenanlagen                              1
Sanierung Gehweg                                                      1
Sanierung Altbau                                                      1
Gehwegsanierung und Verkehrsberuhigung                                1
Gehwegsanierung                                                       1
Gebäudesanierung                                                      1
Sanierung TWL und Einflechtung Mischwasserkanal                       1
Baustelleneinrichtung für Sanierung Kirrchplatz 10                    1
Sanierung Geh- und Radweg                                             1
Sanierung Mischwasserkanal                                            1
Sanierung Gebäude                                                     1
Sanierung Trinkwasserleitung 3. BA                                    1
Sanierung Wohnhaus                                                    1
Kabelsanierung                                                        1
Sanierung Schachtdeckel                                               1
Sanierung Regen- und Trinkwasserleitung                               1
Sanierung/Neubau Hausanschluss                                        1
Gerüststellung & BE für Fassadensanierung                             1
Haussanierung (Gerüststellung, BE entlang des Hauses und Lieferzone)  1
Sanierung Banhunterweg/Erneuerung Stromwandler                        1
Sanierung Wasser/ Abwasser in versch. Bauphasen                       1
Grundhafte Sanierung in 2 Abschnitten, Herstellung Bushaltestellen    1
Name: count, dtype: int64
```

Bitte beachten Sie: Zuerst müssen Sie das Attribut `.str` verwenden, um eine Spalte für Zeichenfolgenoperationen zu öffnen. `.cabin.contains()` funktioniert nicht. `.contains` gibt einen logischen Wert zurück oder NA im Falle eines fehlenden Eintrags.

Ersetzen von Zeichenfolgen

Zuletzt wollen wir im Datensatz die deutschen Umlaute ersetzen, z.B. weil diese Probleme mit ML-Modellen bereiten könnten. Dies kann mit der Methode `str.replace` gemacht werden. Sie hat zwei Argumente: Muster und Ersatz. Wir geben das erstes einen Umlaut wie ä an und ersetzen ihn einfach durch ae:

```
baust.sparte.str.replace("ä", "ae").str.replace("ö", "oe").str.replace("ü",
"ue").str.replace("ß", "ss").value_counts()
```

```
sparte
Hochbau               40
Gebaeudesanierung     35
Strassenbau           35
Wasserleitung         25
```

```
Fernwaermeleitung      18
Telefonnetz            16
Kabelnetz              11
Stromnetz               8
Gruenpflege             4
Lichtsignalanlage       3
Strassenbeleuchtung     3
Gleisbau                2
Kranarbeiten            2
Gasleitung              1
privat                  1
Name: count, dtype: int64
```

In praktischer Hinsicht ist es oft nützlich, den Originalspalte zu behalten, um zu prüfen, ob die Ersetzung richtig durchgeführt wurde. Dies kann erreicht werden, indem eine neue Spalte im Dataframe erstellt wird und eine Stichprobe der alten und neuen Variablen ausgedruckt wird:

```
baust["sparte_ASCII"] = baust.sparte.str.replace("ä", "ae").str.replace("ö",
"oe").str.replace("ü", "ue").str.replace("ß", "ss")
baust[["sparte", "sparte_ASCII"]].value_counts()
```

```
sparte              sparte_ASCII
Hochbau             Hochbau                40
Gebäudesanierung    Gebaeudesanierung      35
Straßenbau          Strassenbau            35
Wasserleitung       Wasserleitung          25
Fernwärmeleitung    Fernwaermeleitung      18
Telefonnetz         Telefonnetz            16
Kabelnetz           Kabelnetz              11
Stromnetz           Stromnetz               8
Grünpflege          Gruenpflege             4
Lichtsignalanlage   Lichtsignalanlage       3
Straßenbeleuchtung  Strassenbeleuchtung     3
Gleisbau            Gleisbau                2
Kranarbeiten        Kranarbeiten            2
Gasleitung          Gasleitung              1
privat              privat                  1
Name: count, dtype: int64
```

Umgang mit Zeitstempeln

Wenn man Zeitreihen analysiert, muss man immer mit Zeitstempeln umgehen. Das ist leider fehleranfälliger, als man vermuten mag, weil es sehr viele unterschiedliche Arten gibt Zeitstemmpel darzustellen und es Besonderheiten wie Zeitzonen, Sommer/Winter-Zeit, oder Schaltjahre gibt.

Prinzipiell können wir Datumsangaben und Zeiten mit der Funktion `pd.to_datetime` in ein richtiges Datetime-Objekt umwandeln, mit dem man auch korrekte Zeitberechnungen machen

kann (und nicht solche Workarounds, wie oben). Bei einzelnen Werten kann die Funktion sogar Datums und Zeitformate selbst erkennen.

```
pd.to_datetime("2023-03-08")
```

```
Timestamp('2023-03-08 00:00:00')
```

```
pd.to_datetime("03/09/2023")
```

```
Timestamp('2023-03-09 00:00:00')
```

```
pd.to_datetime("10-Mar-2023")
```

```
Timestamp('2023-03-10 00:00:00')
```

```
pd.to_datetime("2023-03-11")
```

```
Timestamp('2023-03-11 00:00:00')
```

```
pd.to_datetime("12:00")
```

```
Timestamp('2026-05-11 12:00:00')
```

```
pd.to_datetime("14:30:00")
```

```
Timestamp('2026-05-11 14:30:00')
```

```
pd.to_datetime("10:00:00.400")
```

```
Timestamp('2026-05-11 10:00:00.400000')
```

```
pd.to_datetime("18:00+0030")
```

```
Timestamp('2026-05-11 18:00:00+0030', tz='UTC+00:30')
```

```
pd.to_datetime("2023-03-08 12:00")
```

```
Timestamp('2023-03-08 12:00:00')
```

```
pd.to_datetime("03/09/2023 14:30:00")
```

```
Timestamp('2023-03-09 14:30:00')
```

```
pd.to_datetime("10-Mar-2023 10:00:00.400")
```

```
Timestamp('2023-03-10 10:00:00.400000')
```

```
pd.to_datetime("2023-03-11T18:00+0030")
```

```
Timestamp('2023-03-11 18:00:00+0030', tz='UTC+00:30')
```

Wenn wir diese allerdings in einer Serie kombinieren, funktioniert das nicht mehr

```
# Beispieldaten
data = pd.DataFrame({
    "Datumzeit": ["2023-03-08 12:00","03/09/2023 14:30:00","10-Mar-2023
10:00:00.400","2023-03-11T18:00+0030"]
})
```

```
# Spalte "Uhrzeit" in Zeitformat konvertieren
data["DatumzeitDT"] = pd.to_datetime(data["Datumzeit"])
```

```
ValueError: time data "03/09/2023 14:30:00" doesn't match format "%Y-%m-%d %H:
%M". You might want to try:
    - passing `format` if your strings have a consistent format;
    - passing `format='ISO8601'` if your strings are all ISO8601 but not
necessarily in exactly the same format;
    - passing `format='mixed'`, and the format will be inferred for each
element individually. You might want to use `dayfirst` alongside this.
[31m---------------------------------------------------------------------------[39m
[31mValueError[39m                                Traceback (most recent
call last)
[36mCell[39m[36m [39m[32mIn[47][39m[32m, line 2[39m
[32m      1[39m [38;5;66;03m# Spalte "Uhrzeit" in Zeitformat
konvertieren[39;00m
[32m----> [39m[32m2[39m data[[33m"DatumzeitDT"[39m] =
pd.to_datetime(data[[33m"Datumzeit"[39m])
```

```
[36mFile [39m[32m~/Documents/code/Lehre/
IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/
tools/datetimes.py:1040[39m, in [36mto_datetime[39m[34m(arg, errors,
dayfirst, yearfirst, utc, format, exact, unit, origin, cache)[39m
[32m   1038[39m         result = arg.map(cache_array)
[32m   1039[39m     [38;5;28;01melse[39;00m:
[32m-> [39m[32m1040[39m         values =
[43mconvert_listlike[49m[43m([49m[43marg[49m[43m.
[49m[43m_values[49m[43m,[49m[43m
[49m[38;5;28;43mformat[39;49m[43m)[49m
[32m   1041[39m         result = arg._constructor(values, index=arg.index,
name=arg.name)
[32m   1042[39m [38;5;28;01melif[39;00m
[38;5;28misinstance[39m(arg, (ABCDataFrame, abc.MutableMapping)):

[36mFile [39m[32m~/Documents/code/Lehre/
IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/
tools/datetimes.py:435[39m, in
[36m_convert_listlike_datetimes[39m[34m(arg, format, name, utc, unit,
errors, dayfirst, yearfirst, exact)[39m
[32m    433[39m [38;5;66;03m# `format` could be inferred, or user didn't
ask for mixed-format parsing.[39;00m
[32m    434[39m [38;5;28;01mif[39;00m [38;5;28mformat[39m
[38;5;129;01mis[39;00m [38;5;129;01mnot[39;00m
[38;5;28;01mNone[39;00m [38;5;129;01mand[39;00m
[38;5;28mformat[39m != [33m"[39m[33mmixed[39m[33m"[39m:
[32m--> [39m[32m435[39m     [38;5;28;01mreturn[39;00m
[43m_array_strptime_with_fallback[49m[43m([49m[43marg[49m[43m,[49m
[49m[43m [49m[43mname[49m[43m,[49m[43m
[49m[43mutc[49m[43m,[49m[43m
[49m[38;5;28;43mformat[39;49m[43m,[49m[43m
[49m[43mexact[49m[43m,[49m[43m [49m[43merrors[49m[43m)[49m
[32m    437[39m result, tz_parsed = objects_to_datetime64(
[32m    438[39m     arg,
[32m    439[39m     dayfirst=dayfirst,
[32m   (...)[39m[32m    443[39m
allow_object=[38;5;28;01mTrue[39;00m,
[32m    444[39m )
[32m    446[39m [38;5;28;01mif[39;00m tz_parsed
[38;5;129;01mis[39;00m [38;5;129;01mnot[39;00m
[38;5;28;01mNone[39;00m:
[32m    447[39m     [38;5;66;03m# We can take a shortcut since the
```

```
datetime64 numpy array[39;00m
[32m    448[39m     [38;5;66;03m# is in UTC[39;00m

[36mFile [39m[32m~/Documents/code/Lehre/
IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/
tools/datetimes.py:470[39m, in
[36m_array_strptime_with_fallback[39m[34m(arg, name, utc, fmt, exact,
errors)[39m
[32m    459[39m [38;5;28;01mdef[39;00m[38;5;250m
[39m[34m_array_strptime_with_fallback[39m(
[32m    460[39m     arg,
[32m    461[39m     name,
[32m   (...)[39m[32m    465[39m     errors: [38;5;28mstr[39m,
[32m    466[39m ) -> Index:
[32m    467[39m [38;5;250m    [39m[33;03m"""[39;00m
[32m    468[39m [33;03m    Call array_strptime, with fallback behavior
depending on 'errors'.[39;00m
[32m    469[39m [33;03m    """[39;00m
[32m--> [39m[32m470[39m     result, tz_out =
[43marray_strptime[49m[43m([49m[43marg[49m[43m,[49m[43m
[49m[43mfmt[49m[43m,[49m[43m
[49m[43mexact[49m[43m=[49m[43mexact[49m[43m,[49m[43m
[49m[43merrors[49m[43m=[49m[43merrors[49m[43m,[49m[43m
[49m[43mutc[49m[43m=[49m[43mutc[49m[43m)[49m
[32m    471[39m     [38;5;28;01mif[39;00m tz_out
[38;5;129;01mis[39;00m [38;5;129;01mnot[39;00m
[38;5;28;01mNone[39;00m:
[32m    472[39m         unit = np.datetime_data(result.dtype)
[[32m0[39m]

[36mFile [39m[32mpandas/_libs/tslibs/strptime.pyx:563[39m, in
[36mpandas._libs.tslibs.strptime.array_strptime[39m[34m()[39m
[32m--> [39m[32m563[39m [33m'Could not get source, probably due
dynamically evaluated source code.'[39m

[36mFile [39m[32mpandas/_libs/tslibs/strptime.pyx:511[39m, in
[36mpandas._libs.tslibs.strptime.array_strptime[39m[34m()[39m
[32m--> [39m[32m511[39m [33m'Could not get source, probably due
dynamically evaluated source code.'[39m

[36mFile [39m[32mpandas/_libs/tslibs/strptime.pyx:617[39m, in
[36mpandas._libs.tslibs.strptime._parse_with_format[39m[34m()[39m
[32m--> [39m[32m617[39m [33m'Could not get source, probably due
```

```
dynamically evaluated source code.'[?[39m

[?[31mValueError[?[39m: time data "03/09/2023 14:30:00" doesn't match format
"%Y-%m-%d %H:%M". You might want to try:
    - passing `format` if your strings have a consistent format;
    - passing `format='ISO8601'` if your strings are all ISO8601 but not
necessarily in exactly the same format;
    - passing `format='mixed'`, and the format will be inferred for each
element individually. You might want to use `dayfirst` alongside this.
```

Deshalb muss man zuerst darauf achten, dass entsprechende Zeitstempel immer in einem festen Format sind.

```
# Beispieldaten
data = pd.DataFrame({
    "Datumzeit": [
        "2023-03-08T12:00:00.000+0030",
        "2023-03-09T14:30:00.000+0030",
        "2023-03-10T10:00:00.400+0030",
        "2023-03-11T18:00:00.000+0030"]
})
```

```
# Spalte "Uhrzeit" in Zeitformat konvertieren
data["DatumzeitDT"] = pd.to_datetime(data["Datumzeit"], format='ISO8601')
data.DatumzeitDT
```

```
0          2023-03-08 12:00:00+00:30
1          2023-03-09 14:30:00+00:30
2   2023-03-10 10:00:00.400000+00:30
3          2023-03-11 18:00:00+00:30
Name: DatumzeitDT, dtype: datetime64[us, UTC+00:30]
```

Bei unsicheren Mustern kann man das Format direkt spezifizieren, Dafür wird die strftime-Notation verwendet

```
# Spalte "Uhrzeit" in Zeitformat konvertieren
data["DatumzeitDT"] = pd.to_datetime(data["Datumzeit"], format="%Y-%m-%dT%H:
%M:%S.%f%z")
data.DatumzeitDT
```

```
0          2023-03-08 12:00:00+00:30
1          2023-03-09 14:30:00+00:30
2   2023-03-10 10:00:00.400000+00:30
```

```
3          2023-03-11 18:00:00+00:30
Name: DatumzeitDT, dtype: datetime64[us, UTC+00:30]
```

Mit dem gleichen Muster können Datumsangaben auch wieder in Strings umgewandelt werden.

```
data.DatumzeitDT.dt.strftime('%Y-%m-%d %H:%M')
```

```
0    2023-03-08 12:00
1    2023-03-09 14:30
2    2023-03-10 10:00
3    2023-03-11 18:00
Name: DatumzeitDT, dtype: str
```

Bibliographie